Software Architecture in DoD Acquisition: An Approach and Language for a Software Development Plan

John K. Bergey
Paul C. Clements

February 2005

Software Architecture Technology Initiative

Technical Note
CMU/SEI-2005-TN-019

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

Acknowledgments

We thank Sholom Cohen, Lawrence Jones, and Linda Northrop for their careful reviews.

About the Technical Note Series on Software Architecture Practices in the Department of Defense

The Product Line Systems Program at the Carnegie Mellon® Software Engineering Institute (SEI) is publishing a series of technical notes designed to condense knowledge about software architecture practices into a concise and usable form for the Department of Defense (DoD) acquisition manager and practitioner. Our objective is to provide practical guidance and lay a conceptual foundation for DoD architecture practice. This series, called Software Architecture Practices in the Department of Defense, is a companion to the SEI series on product line acquisition and business practices [Campbell 02, Bergey 01, Cohen 01, Bergey 00a, Bergey 00b, Jones 99, Bergey 99a].

This technical note is part of a special series of notes titled "Software Architecture in DoD Acquisition" that is aimed at DoD acquisition specialists who are commissioning large software-intensive systems for the DoD. The intent of the series is to explain how to bring the concepts of software architecture effectively into the system acquisition process. Titles currently in the series include

- *A Reference Standard for a Software Architecture Document*: This technical note suggests the layout and contents of each section of a Software Architecture Document [Bergey 05].
- *An Approach and Language for a Software Development Plan*: This technical note offers an example approach and corresponding language that covers software architecture practices and that could be inserted into a contractor's software development plan (SDP). [This note is the one you are now reading.]

Possible future titles include

- *Reviewing a Software Architecture Document*: This technical note will provide a step-by-step approach for the peer review of software architecture documentation, including specific questions and a methodological basis for achieving high-quality reviews.
- *An Approach and Language for a Software Architecture Evaluation Plan*: This technical note will offer an example approach and corresponding language for creating a plan to conduct a series of in situ software architecture evaluations in a system acquisition using the Architecture Tradeoff Analysis Method® (ATAM®).

® Carnegie Mellon, Architecture Tradeoff Analysis Method, and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

- *A Reference Standard for a Software Architecture Evaluation Report*: This technical note will suggest the layout and contents of each section of an ATAM evaluation report.

Abstract

The right software architecture is essential for a software-intensive system. Meeting behavioral requirements and providing quality attributes such as real-time performance, reliability, and maintainability are essential architectural drivers. Because an architecture comprises the earliest, most important, and most far-reaching design decisions, making sure that the architecture will be fit for purpose is one of the most powerful, technical risk mitigation strategies available to a program office. This technical note covers one avenue of exercising architectural control—the Software Development Plan (SDP). The report provides an example approach and corresponding SDP language that enable software architecture to play a central role in the technical and organizational management of a software development effort. The example is drawn from an actual SDP written by a major U.S. Department of Defense contractor in a weapon-system procurement. The intent is to provide an example for other acquisition organizations to use (and adapt as appropriate) in their own procurements. While the example is based on a contracting approach with a lead system integrator, it can serve as a model for using an architecture-centric approach effectively to unify and manage software development across multiple suppliers, as found in the conventional prime-with-subcontractors acquisition context.

1 Introduction

The right software architecture is essential for a software-intensive system to meet its behavioral requirements as well as its obligations to provide quality attributes such as real-time performance, reliability, and maintainability. Software architecture is especially critical in large, complex systems. Assuring that the right architecture is being developed is difficult enough in the context of in-house development, but an acquisition organization has an even harder task because its contact and leverage points with the contractor(s) are limited, occur at discrete points in the life cycle, and are exercised from a distance.

Consequently, it is important for an acquisition organization that commissions the development of large software systems to exercise its oversight prerogatives with respect to software architecture. Such an organization is the U. S. Department of Defense (DoD). Because an architecture comprises the earliest, most important, and most far-reaching design decisions, making sure that the architecture will be fit for purpose is one of the most powerful technical risk mitigation strategies available to a DoD program office.

Bergey covers the contracting mechanisms that can be brought to bear by DoD procurement agencies to help assure the development and delivery of a sound architecture that will serve the program throughout its entire life cycle [Bergey 99b].

This technical note covers another avenue of exercising architectural control—the Software Development Plan (SDP). The SDP lays out the guiding principles, practices, and guidelines that members of the software development team must follow in order to produce software that meets its requirements in a cost-effective way.

Many team structures are possible, but the same principles and needs apply in every case where a *team* cooperates to produce a product whose technical foundation is its architecture. The context for the approach being described involves having a system prime contractor (under the auspices of a DoD program office) serve as a lead system integrator (LSI). The LSI, in turn, is responsible for contracting with multiple suppliers to develop (or otherwise acquire) the software that will be part of a new “system of systems (SoS)” being developed to satisfy the DoD’s mission needs. Creating an overarching, architecture-centric SDP provides the LSI with an effective means for unifying, guiding, and managing the entire software development effort across multiple suppliers. Moreover, such unification, guidance, and management are also needed in the more conventional acquisition context in which a prime contractor brings a number of subcontractors aboard. Even in the case where groups of people within a single contractor organization perform the work, these same principles apply.

This SDP typically establishes the program-level framework and requirements for processes used in the acquisition, development, integration, test, and support of the system software that is being acquired ultimately from multiple suppliers. It also establishes the top-level software development and integration plan for the software. In an acquisition featuring a multilevel work structure (such as a prime contractor and subcontractors, or an LSI and suppliers), the SDP imposes binding process requirements at all levels. It may call for subcontractors or suppliers to produce their own SDPs that must comply with the SDP of the prime contractor or the LSI.

The purpose of this technical note is to provide an example of an effective approach and corresponding language for an SDP that will enable software architecture to play a central role in the technical and organizational management of a software development effort. In this way, a program office can have something to compare to an SDP provided by its contractor or contractors. Alternatively, the program office can adapt this approach and corresponding language to reflect the kind of issues it wants its contractor's SDP to address. Of course, neither the comparison or the adaptation will be effective unless the acquisition agency contractually requires the creation and delivery of, and adherence to, an SDP.

A companion to this technical note offers an example of a reference standard that prescribes the contents and organization of a software architecture documentation package [Bergey 05]. This type of documentation is a prerequisite for conducting a software architecture evaluation.

2 An Example SDP

This technical note provides an example approach and corresponding language to enable software architecture to play a central role in an SDP. The approach and language were drawn from (but are not identical to) that used by a major contractor in a DoD weapon-system acquisition. The intent is to provide an example for other acquisition organizations to use (and adapt as appropriate) in their own procurements.

2.1 Context of the Example

In the particular example we've chosen to illustrate (Figure 1), the prime contractor is known as the *LSI* whose primary job is to manage and oversee the integration of large segments of software provided (produced, procured) by *segment teams*. Segment teams in turn contract with *suppliers* to provide particular units of software. The result of the LSI's integration efforts is an *SoS*, in which software plays a critical role.

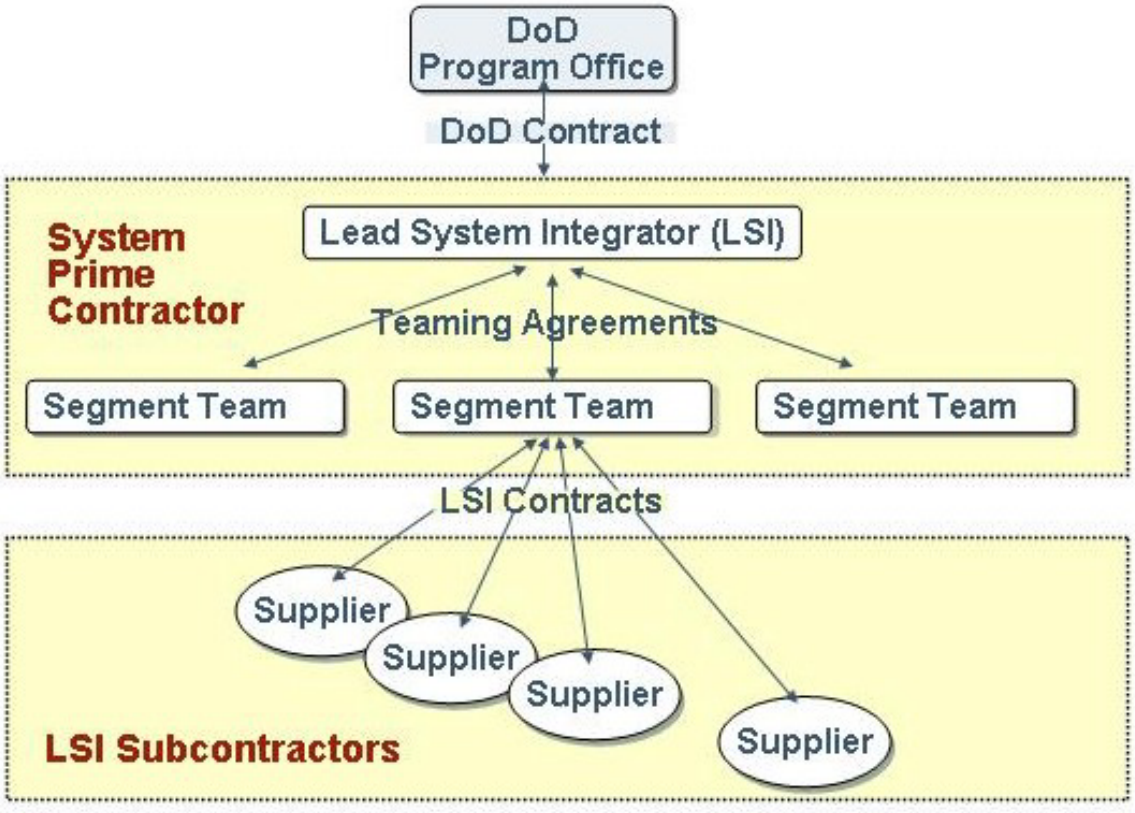

Figure 1: Example System Acquisition Environment

The LSI is responsible for the *SoS software architecture*, which assures that the segments will integrate smoothly and that the final product will meet its behavioral and quality goals. Segment teams are responsible for architecting and developing the segments in accordance with the SoS software architecture.

The LSI, each segment team, and each supplier providing a subsystem with its own software architecture all are obligated to appoint a chief software architect whose duties are spelled out in the SDP. They also are required to appoint a software architecture team as needed to support the software architect in his or her duties.

2.2 Making the Example Work in Other Contexts

The LSI context involves the creation of a collection of software architecture documents and related artifacts, and the SDP includes language to assure the consistency of those artifacts with each other. In acquisition contexts in which only one software architecture is produced, this language can, of course, be omitted.

For acquisitions involving a simple prime contactor and subcontractors, the material relating to segment teams and suppliers in this example can be replaced by corresponding material relating to subcontractors.

An SDP, like most project documents, should evolve over the life of the project as processes are improved, added, or discarded through experience. The approach and corresponding language presented in this technical note are not intended to be canonical or even generic. They are intended to serve as examples that can inspire alternative approaches and language specific to other acquisition-based software development projects.

2.3 Contents Overview

We present the contents and organization of our example SDP below by summarizing its table of contents. Other SDPs may have different contents and organizations; this one is only an example, but it typifies the information content of SDPs for large software-intensive systems acquired by the DoD. Our example SDP does not mandate a particular notation (such as UML) for capturing architectural information. Choice of notation should be covered in an SDP—by either mandating a choice or levying notation-selection criteria—but we have omitted this issue for the purposes of this technical note, so as not to appear to endorse one notation over another.

The example SDP comprises the following chapters:

- Chapter 1 – Scope: program overview, an overview of the application, an overview of the SDP, its relationship to other program plans, and its relationship and authority over lower level supplier SDPs
- Chapter 2 – Applicable Documents: other documents that are referenced in the SDP and whether they are informational or compliance documents
- Chapter 3 – Overview of Required Work: overview of the application from an operational point of view and an overview of the computing platform configuration envisioned to support the application; support software; and project constraints and overarching requirements, including cost and schedule requirements
- Chapter 4 – Software Engineering: the heart of the SDP; description of the software life-cycle model (in the case of our example, Boehm's Spiral Win-Win Model [USC 05]); description of build strategies and identification of a build plan; description of strategies for preplanned software reuse and a product line approach to software development; description of the software architecture's role in the project; and how the architecture will be captured, documented, modeled, and evaluated
  - 4.1 – Software Development: life-cycle model, build strategy, build plans and schedules, documentation requirements and constraints
  - 4.2 – Software Reuse: product line strategy, strategies for qualification, rehabilitation and reuse, and integration of existing software
  - 4.3 – Software Architecture Development: modeling, review, evaluation, documentation, and notation for software architecture
  - 4.4 – Software Development Methods
  - 4.5 – Standards for Software Products
  - 4.6 – Handling of Critical Requirements: safety-critical requirements, mission-critical requirements, information assurance requirements
  - 4.7 – Computer Hardware Resource Allocation
  - 4.8 – System Requirements Analysis: specifications, requirements decomposition and traceability, flow-down
  - 4.10 – System Design
  - 4.11 – Software Requirements Analysis
  - 4.12 – Software Design: object-oriented development, use of a notation (e.g., UML), design exit criteria
  - 4.13 – Software Implementation and Unit Testing
  - 4.14 – Software Verification: unit integration and testing, qualification testing
  - 4.15 – Preparing for Software Use: inspections and preparation of the executable software, version descriptions, user manuals, and software transition

- Chapter 5 – Software Project Management: activities required to support cost and schedule estimation for software, measurement, communication, and coordination among groups (including project repositories and software development folders), software training, software risk management, subcontract management, software configuration management, software product evaluation, software quality assurance, and defect management
- Chapter 6 – Project Organization and Resources: role and structure of software management on the project, the software development environment, and its deployment plan, security considerations, and so forth
- Chapter 7 – Project Software Process Management: process management for the project, including how such processes (including those defined in this SDP) are evaluated and changed

3 Example SDP Approach and Language Pertaining to Software Architecture

This section of the technical note lays out the role of software architecture in the acquisition project (the approach) and presents example language that populates Section 4.3 of the example SDP.

3.1 Example Section 4.3: Software Architecture Development

3.1.1 Example Section 4.3 Introduction

Software architecture plays a central role in the technical and organizational management of the software development effort. A suitable architecture is a prerequisite for satisfying requirements, both in terms of accommodating the needed functionality and in providing the necessary quality attributes such as real-time performance, reliability, and security. Elements developed separately will work together only if they conform to the architecture, which includes precise statements of the elements' interfaces. In addition, a properly defined and maintained architecture increases the amount of strategic reuse achievable in the product line, reducing the cost and schedule of future upgrades to the system.

The software architecture defines the software elements that must be acquired through the development, open-market purchase, or reuse of existing assets. It also determines how effectively the software requirements are met and, to some extent, the ease with which products developed separately can be integrated together. Further, architecture provides the means to develop the software system as a series of incremental builds. By carefully engineering the "uses" relations among elements, the architect can assure that incrementally more useful subsets of the entire software system can be developed, tested, and deployed separately.

The actors who participate in the architecture aspects of the SDP and the key architecture artifacts involved are summarized in Figure 2.

Actors	Software Architecture for …	Architecture Artifacts			
		Documents		Plans	
		Software Architecture Document	Architecture Evaluation Report	Architecture Evolution Plan	Architecture Evaluation Plan
LEAD SYSTEM INTEGRATOR Chief Software Architect and Software Architecture Team	System of Systems	Produce	Produce	Produce	Produce
	Segment	Review and approve	Review and approve	Review and approve	Review and approve
	Key Subsystems	Review and approve	Review and approve	Review and approve	Review and approve
SEGMENT TEAM Chief Software Architect and Software Architecture Team	System of Systems	Help LSI produce; review and conform	Help LSI produce; review and conform	Help LSI produce; review and conform	Help LSI produce; review and conform
	Segment	Produce	Produce	Produce	Produce
	Key Subsystems	Review and approve	Review and approve	Review and approve	Review and approve
SUPPLIER Chief Software Architect and Software Architecture Team	System of Systems	Review and conform	Review and conform	Review and conform	Review and conform
	Segment	Review and conform	Review and conform	Review and conform	Review and conform
	Key Subsystems	Produce	Produce	Produce	Produce

Figure 2: Overview of Actor Responsibilities for Key Architecture Artifacts

The responsibilities of these actors for the various architecture artifacts that fall under the scope of the SDP are elaborated in the sections that follow.

3.1.2 Example Section 4.3.1 Responsibilities Pertaining to Software Architecture

The LSI (who develops the overall SoS software architecture), segment teams (who develop software architectures for project segments) and suppliers (who develop software for segments that must be in accordance with the segment and SoS software architectures, and which may be complex enough to warrant its own software architecture) shall perform the following actions:

- Appoint a chief software architect and software architecture team for the software architectures they are responsible for delivering.
- Define the software architecture to satisfy the project's functional and quality attribute requirements, as well as its incremental build requirements.

- Define the incremental builds that the architecture needs to support.
- Conduct one or more evaluations of the architecture (or, if appropriate, of the architecture of key subsystems), using a method such as the SEI Architecture Tradeoff Analysis Method® (ATAM®) [Clements 01]. An evaluation requires suppliers to produce and deliver an architecture evaluation plan[1] and evaluation report. The plan addresses such items as the scheduling and phasing of evaluations; the method being used; evaluation team training; the tailoring of the method; the artifacts involved; the roles and responsibilities of DoD, LSI, and supplier participants; the deliverables; and the use of evaluation results.
- Produce a software architecture document, in the format promulgated by the LSI's chief software architect (unless a waiver has been granted by the chief software architect).
- Produce an architecture evolution plan. The architecture evolution plan specifies who is in charge of the architecture over its lifetime and how changes to the architecture are proposed and handled. If an increment is downsized to fit into the schedule, the architecture evolution plan is updated to communicate the deferred requirements to suppliers as evolution requirements that their architecture needs to support.

The LSI's chief software architect has project responsibility and authority for the following:

- leading the LSI's software architecture team
- defining the software architecture document format[2]
- producing a baseline set of architecture viewpoints, as defined by the American National Standards Institute/Institute of Electrical and Electronics Engineers (ANSI/IEEE) that shall be considered when producing a software architecture document [IEEE 00]
- defining and maintaining appropriate processes for all life-cycle aspects of the software architecture consistent with the requirements of this SDP, over the lifetime of the program
- producing an SoS software architecture document, in accordance with the requirements of this SDP. The review team for the software architecture document shall include segment teams and supplier software architects. This document shall be maintained by the LSI's chief software architect over the life of the program.
- approving waiver requests for segment teams and suppliers involving changes to the contents of, or the process for, the software architecture document, as required by this SDP and published software architecture process guidance
- reviewing software architecture documents created by segment teams and suppliers (as required by this SDP and published software architecture process guidance) with

® Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1 An example software architecture evaluation plan will be described in a future technical note.

2 Bergey and Clements provide guidance for a software architecture document [Bergey 05].

particular focus on how the software architecture being described fits into the overall system

- approving software architecture documents for segment teams and suppliers, as required by this SDP

Each segment team and supplier shall be responsible for the following:

- appointing a chief software architect, who shall lead the production of the software architecture artifacts for the segment teams and suppliers. Chief software architects of the segment teams are members of the LSI's software architecture team. Software architects for major suppliers (as designated by the segment team or LSI chief software architect) also are members of the LSI software architecture team and perform reviews of the SoS software architecture document.
- producing the architecture evolution plan and software architecture document, as provided by this SDP and published software architecture process guidance
- ensuring that the software produced by the segment teams and suppliers conforms to the published software architecture document for these groups
- ensuring that the software architecture document and related software architecture artifacts (e.g., architecture evolution plan and software architecture process guidance) are maintained over the lifetime of the program or the supplier's contract

3.1.3 Example Section 4.3.2 Software Architecture Viewpoints

The program employs a stakeholder-focused, multiple-view approach to architecture documentation. A *viewpoint* (following the terminology of ANSI/IEEE) is "a specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 00]." A *view,* which conforms to a viewpoint, is the description of a specific architecture, defined to meet the requirements of its associated viewpoint. The viewpoint identifies the set of concerns to be addressed and the modeling techniques, evaluation techniques, consistency-checking techniques, and so forth used by any conforming view. A view is a representation of a set of system elements and the relationships among them, as specified by the view's associated viewpoint. Together, the chosen set of views shows the entire architecture and all of its relevant properties. Software architecture documents contain the viewpoints, relevant views, and information that apply to more than one view to give a holistic view of the system. Software architecture documentation involves the following steps:

1. The LSI chief software architect, in conjunction with stakeholder participants, produces an initial set of viewpoints (hereafter referred to as predefined viewpoints). From this set, the LSI chief software architect chooses the viewpoints to be used in the SoS software architecture document. The LSI chief software architect denotes some of these predefined viewpoints as mandatory, meaning that every software architecture document shall include these viewpoints. The other viewpoints are marked as suggested.

2. Each segment team or supplier chief software architect identifies the set of relevant viewpoints for the specific software system or subsystem based on specific project needs as articulated by the stakeholders of the architecture. To define the viewpoints, the software architect

   - produces a stakeholder/viewpoint table, an example of which is shown in Figure 3. The rows indicate stakeholders of the architecture documentation, and the columns represent viewpoints that apply to the subject of the architecture (the entire set of software or some portion thereof).
   - examines the set of predefined viewpoints. The software architect may combine viewpoints to reduce documentation overhead while still meeting stakeholder needs. The software architect may also add viewpoints, as necessary, to meet specific stakeholder concerns not satisfied by the predefined viewpoint set.
   - prioritizes and makes the final selection from the viewpoints emerging from the examination of the candidate views. The software architect shall document the final viewpoint selection, including rationale for any predefined viewpoints not used in this software architecture document. The software architect also shall update the stakeholder/viewpoint table, to show how the final set of viewpoints meets all the stakeholders' concerns.

The software architect documents the selected views, as defined by the viewpoints and the guidance in the next section. Processes, techniques, and templates for selecting and documenting software architectural views are provided by IEEE [IEEE 00] and Clements and colleagues [Clements 02].

	Viewpoints							
Stakeholder	Logical	Process	Deploy-ment	Implemen-tation	Use Case	Security	Data	Systems Mgmt
Application SW Developer	P	P	S	P	P	S	P	S
Infrastructure SW Developers	P	P	P	P	S	P	P	P
Application System Engineers	P	S	P		P	P	S	S
Application/Platform Hardware Engineers	S	P	S		S			S
Security Engineers/Certifiers	S	P	P	S	S	P	S	P
Safety Engineers/Certifiers	S	S	S	P	P	S	S	
Reliability Engineers/Certifiers	S	P	P				S	P
Quality Assurance Engineers	P	S		P	S	S	S	
Communications Engineers	S	P	P		S	S		S
System of Systems Engineers	S	S	P		P	P	S	S
LSI Program Management	S			P	S			
Business Management	S			P	S			
System Integration and Test Engineers	P	S	P	S	P	S	S	P
External Test Agencies	S	S	P	S	P	S	S	P
Govt. Program Management	S	S	S	P	S	S	S	S
End Users (commanders, soldiers, etc)		S	P			P	S	S
Operational Systems Managers		S	S			S	P	P
Trainers		S	S		P		S	S
Maintainers	P	P	S	P	S	S	P	S
Other Services			S		S	S	P	S
Auditors				P				
Chief Engineer/Chief Scientist	P	S	S	S	S	S	P	S
Other Service Commands	P		P		P	P	P	S
Service Acquisition/Policy		S	S	P		S	P	P
Representatives of standardization activities	P	P	S	S			P	

(P = primary concern, S = secondary concern)

Figure 3: Example Stakeholder/Viewpoint Table

3.1.4 Example Section 4.3.3 Software Architecture Document

For segment teams and suppliers that generate software architecture, all software architectures shall be documented in a software architecture document. The SoS software architecture document shall be the governing software architecture document, with all other software architecture documents conforming to its requirements and precepts.

The target audience of the software architecture document includes system architects and engineers as well as software architects and engineers. The software architecture document should be concise and accessible to the newcomer and serve as a reference for everyone involved.

Each software architecture document serves as the authoritative source of architectural information for its architectural scope. Software architecture documents are further intended to be the starting point for locating all software architecture information on the program. Software architecture documents are both descriptive (what the system or software component should look like) and prescriptive (how the software component should be designed or how it behaves).

Contents shall conform to the requirements for documentation defined by the LSI chief software architect. Documentation of a view includes a primary presentation (often graphical), an element catalog that describes the elements shown in the primary presentation

(including their interface specifications), and other supporting information. In general, views that correspond to the predefined viewpoints will conform to the full requirements of that viewpoint. If a software architect desires to use an alternate viewpoint or other aspect in a view, the software architect shall obtain prior approval from the LSI chief software architect. The software architect shall, in addition to views, provide information that applies to more than one view, including a mapping between views to show the relationships among elements in different views, a documentation roadmap to help a stakeholder find information in the package relevant to his or her interests, and other supporting information.

The software architect shall conduct reviews on the resulting software architecture document. The software architect shall obtain approval for the software architecture document, as described in the next section.

3.1.5 Example Section 4.3.4 Software Architecture Products Review and Approval

Software architecture artifacts (e.g., software architecture document and architecture evolution plan) shall be formally reviewed. Some additional requirements apply to reviews of software architecture documents by the LSI software architecture team:

- Waiver requests (e.g., alternate viewpoints, alternate modeling methods, and evaluation approaches) shall be submitted and approved by the LSI chief software architect prior to review by the LSI software architecture team.
- The software architect (segment team or supplier) shall conduct a review of the software architecture *defining material* prior to developing the views in the software architecture document. The defining material covered by the review shall include
  - a list of the stakeholders and their concerns
  - the selected viewpoints and the rationale for the selection
  - a list of the critical reviewers and a mapping back to the architecture stakeholders
  - the schedule for developing, reviewing, and evaluating the software architecture document
- The software architect (segment team or supplier) shall conduct one or more reviews of the full software architecture document with particular focus on the contents of the actual architectural views.
- The software architect may conduct formal evaluations of the document, once it has completed peer reviews, as described in the next section.
- Once all reviews and evaluations are complete, the software architect shall submit the software architecture document to the chief software architect for approval.

3.1.6 Example Section 4.3.5 Software Architecture Evaluation

The LSI performs architecture evaluations to check the SoS software architecture for fitness of purpose by using the ATAM [Clements 01]. The ATAM calls for assembling a small group of system stakeholders and elicits from them a set of scenarios that articulate the system's important required behaviors and quality attribute requirements. Then it investigates how the architecture satisfies each of the most important scenarios.

At a minimum, the LSI evaluates the SoS software architecture for the following criteria:

- allowance for each member of the SoS to meet individual behavioral and quality attribute requirements, including security, high availability and fault tolerance, real-time performance, and others
- support for achieving the aggressive cost/schedule constraints imposed by the program. Architecturally this support means limiting the size of architectural elements, carefully decomposing responsibility and functionality among the elements, and providing simple and robust interfaces to maximize the potential for developmental concurrency.
- allowance for incremental development, integration, and testing. Architecturally this requirement means paying careful attention to the "uses" relation among elements.
- non-susceptibility to unplanned commercial off-the-shelf (COTS) upgrades or COTS elements becoming unsupported
- support for integration and testing. For example, test-time instrumentation mechanisms could be mandated for inclusion in each architectural element.
- support for instance variability in the product line, meaning that almost-alike systems within the project should contain largely the same software, differing only by configuration parameters or the exercise of other architectural variability mechanisms
- support for commonality, meaning that units of procurement should be defined to maximize the creation of common elements across the system, perhaps as opposed to procuring strictly along functional lines

Segment teams and suppliers shall perform their own software architecture evaluations on major subsystems using a method such as the ATAM. Criteria shall include, but not be limited to, those listed above. Segment teams and suppliers shall document the results of software architecture evaluations in a separate software architecture evaluation report[3] or as part of their overall software architecture documentation. The evaluation report shall include the specific criteria used to evaluate the architecture, the process for gathering the criteria, the process for conducting the evaluation, and the risks and non-risks uncovered about the architecture with respect to each criterion. The ATAM is described fully by Clements and colleagues [Clements 01].

[3] A separate evaluation report is considered preferable because the report typically would be a required deliverable that is produced downstream after the software architecture document has been developed and delivered, and the architecture has been evaluated.

References

URLs are valid as of the publication date of this document.

[Bergey 99a] Bergey, J.; Fisher, M.; & Jones, L. *The DoD Acquisition Environment and Software Product Lines* (CMU/SEI-99-TN-004, ADA244787). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. http://www.sei.cmu.edu/publications/documents/99.reports/99tn004/99tn004abstract.html

[Bergey 99b] Bergey, J. & Fisher, M. *Software Architecture Evaluation with ATAM in the DoD System Acquisition Context* (CMU/SEI-99-TN-012, ADA377450). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. http://www.sei.cmu.edu/publications/documents/99.reports/99tn012/99tn012abstract.html

[Bergey 00a] Bergey, J. & Smith, D. *Guidelines for Using OAR Concepts in a DoD Product Line Acquisition Environment* (CMU/SEI-2000-TN-004, ADA377385). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tn004.html

[Bergey 00b] Bergey, J.; Fisher, M.; Gallagher, B.; Jones, L.; & Northrop, L. *Basic Concepts of Product Line Practice for the DoD* (CMU/SEI-2000-TN-001, ADA375859). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tn001.html

[Bergey 01] Bergey, J. & Goethert, W. *Developing a Product Line Acquisition Strategy for a DoD Organization: A Case Study* (CMU/SEI-2001-TN-021, ADA395202). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01tn021.html

[Bergey 05] Bergey, John & Clements, P. *Software Architecture in DoD Acquisition: A Reference Standard for a Software Architecture Document* (CMU/SEI-2005-TN-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn020.html

[Campbell 02] Campbell, G. *A Software Product Line Vision for Defense Acquisition* (CMU/SEI-2002-TN-002, ADA403810). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tn002.html

[Clements 01] Clements, P.; Kaman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2001.

[Clements 02] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2002.

[Cohen 01] Cohen, S. *Case Study: Building and Communicating a Business Case for a DoD Product Line* (CMU/SEI-2001-TN-020, ADA395155). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01tn020.html

[IEEE 00] Institute of Electrical and Electronics Engineers. *Recommended Practice for Architectural Description of Software-Intensive Systems* (IEEE Std 1471-2000). New York, NY: Institute of Electrical and Electronics Engineers, 2000.

[Jones 99] Jones, L. *Product Line Acquisition in the DoD: The Promise, The Challenges* (CMU/SEI-99-TN-011, ADA373184). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. http://www.sei.cmu.edu/publications/documents/99.reports/99tn011/99tn011abstract.html

[USC 05] University of Southern California. *WinWin Spiral Model*. http://sunset.usc.edu/research/WINWIN/winwinspiral.html (February 2005)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	February 2005	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Software Architecture in DoD Acquisition: An Approach and Language for a Software Development Plan	F19628-00-C-0003

6. AUTHOR(S)

John K. Bergey, Paul C. Clements

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2005-TN-019

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The right software architecture is essential for a software-intensive system. Meeting behavioral requirements and providing quality attributes such as real-time performance, reliability, and maintainability are essential architectural drivers. Because an architecture comprises the earliest, most important, and most far-reaching design decisions, making sure that the architecture will be fit for purpose is one of the most powerful, technical risk mitigation strategies available to a program office. This technical note covers one avenue of exercising architectural control—the Software Development Plan (SDP). The report provides an example approach and corresponding SDP language that enable software architecture to play a central role in the technical and organizational management of a software development effort. The example is drawn from an actual SDP written by a major U.S. Department of Defense contractor in a weapon-system procurement. The intent is to provide an example for other acquisition organizations to use (and adapt as appropriate) in their own procurements. While the example is based on a contracting approach with a lead system integrator, it can serve as a model for using an architecture-centric approach effectively to unify and manage software development across multiple suppliers, as found in the conventional prime-with-subcontractors acquisition context.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software architecture, architecture documentation, architecture evaluation, architecture in acquisition, software development plan	28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL